Wi-Fi Room Control

Muhammed Khan  
Yousuf Khan  
Chi Shing Poon  
Alejandro Valencia  
David Sigala

Electrical Engineering

The University of Texas at Arlington

Arlington, Texas

[muhammedihs.khan@mavs.uta.edu](mailto:muhammed.khan@mavs.uta.edu)

[yousuf.khan@mavs.uta.edu](mailto:yousuf.khan@mavs.uta.edu)

[chishing.poon@mavs.uta.edu](mailto:chishing.poon@mavs.uta.edu)

[alejandro.valencia@mavs.uta.edu](mailto:alejandro.valencia@mavs.uta.edu)

[david.sigala@mavs.uta.edu](mailto:david.sigala@mavs.uta.edu)

***Abstract*—It is possible that every electronic aspect of this world can be controlled from the palm of one’s hands. This is an especially promising idea considering that if applied to a house, or in this case a room, most things can be controlled with the swipe of a finger. The main objective is to create an “internet of things” where each device is connected together and can then be controlled through a smartphone application. This is a very possible scenario that will be demonstrated in this project. The project will be separated into four parts which would be assigned to different team members. The end project would consist of an application to control certain variables in a room (blinds, lights, temperature, etc.), a hub that will send and receive the signals from and to the application and the hardware, and finally a room that would demonstrate the functionality of this project. The project is planned to be finished by the presentation date.**

***Keywords—****Hub, Humidifier, App, Environment, MSP430, Sensors, Photocell, Android Studio, Control Room, Gantt Chart, HC-05, DHT22, Relays, SDK*

# Introduction

Although at its basis a room is composed of four walls, a floor, and a ceiling, it also has other features. Some of these are things such as blinds, lights, temperature controls, etc. Imagine all these parts individually interconnected through Bluetooth from a mobile device. This project is to showcase that scenario.

Essentially, the goal of this project is to create a device that controls a demo room through an application developed on the android platform. This has been done and showcased by many companies in the past, but the aim is to create a system that can be used in the commercial setting. An application of this would be used in hotel rooms. A user would download the hotel’s app and be able to control aspects of the room from there. Another would be an office space where the manager or boss would like to manage the aspects of that room. Finally, there are factories where the manager would like certain machines to be activated or certain sections of the factory to remain dormant for periods of time.

The way the devices will communicate between the user application and the Control room is through the microcontroller, MSP430. The MSP430 will be used with a HC-05 bluetooth module to allow it to send and receive signals via bluetooth over the same network. These are all part of a bigger component called the Hub. The Hub’s main job is to process the signals that are sent from the other hardware or the application, and decode the signals so it can communicate with every component in this project within a timely manner without little signal loss, if any. However, all the hardware required such as the lights, fans, humidifiers, temperature control, and blinds will be connected through hard-wirings; making it simpler and more organized to interconnect every component and enabling us to find any errors within the circuitry caused by any voltage spikes or any short circuits.

# Background

The world is becoming more and more automated as technology advances [1]. Menial tasks which used to require hours of human labor are now done automatically by highly accurate machines at faster rates than were previously thought improbable. This technological revolution does not need to be limited to manufacturing factories however, as it is much more commonly available. Computers have been personalized, the internet is in many households, and yet there are still parts of our lives that have not been optimized.

One such scenario is in our homes. Many of us still need to do one of two extremes. In the first case one needs to get up and pull the switch for the ceiling fan, turn on the air conditioner, push the switch for the lights, set the humidifiers to make sure it doesn’t get so dry as to cause a sore throat, and by the time said individual sits down they’ve already forgotten they also needed to close the blinds. It’s too much to handle at once. This is made especially clear on days when environmental temperature impacts indoor temperature and all one desires to do is come home to a comfortable habitat. The other case is that each has their own remote control, which in itself is impractical. Having to switch between multiple different remotes is tedious and undesirable. People already have enough remote controls as is, with an average of four within a single living room [2]. Instead, it would be much easier and more efficient to be able to control all of these devices at once.

Another scenario where this is applicable is in hotels. Some hotels, such as the Hilton, choose to control all temperature aspects directly at the expense of their guests [3]. This is done because giving guests complete control over a room can lead to many different incidents, however guests feel it undermines their autonomy. Both parties can be satisfied if there was an automated system in place to control the room locally and through hotel staff.

# BLock Diagram & Design

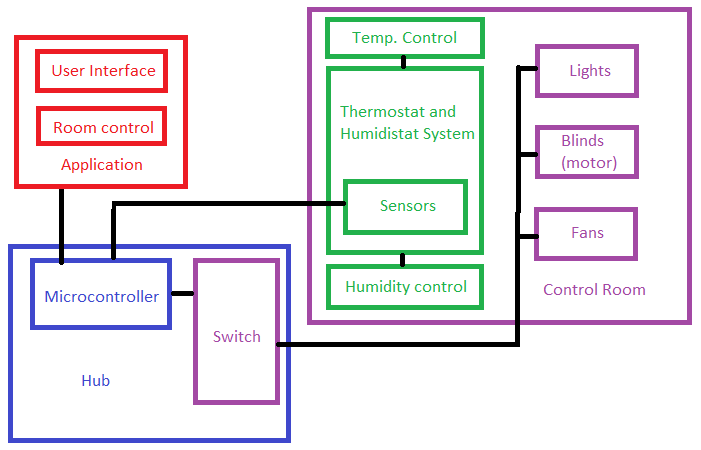
**Figure 1. Block Diagram for Room and Control System**

Figure 1 above shows the block diagram for this project. It illustrates how various components of the project are connected, and what those components are. Since the project is divided into multiple components, the block diagram is color-coded in red, blue, green, and purple for each team member.

The team member’s name and their responsibilities can be found after the block diagram. Where it also includes further explanations on the approach of each component. The original block diagram created at the beginning of the semester still applies to this point. In the hub, the switch was changed. However, the diagram will not change because the switch will still be there in the form of a PCB circuit and relay which will be explained in detail further in this report.

The HUB will be hard-wired to the room via simple jumpers and the compatibility circuit mentioned above. This way devices such as lights, blinds and fans can be controlled through the microcontroller. The room will be plugged directly into a wall socket and can be demonstrated anywhere. When plugged in, the room will provide power to all components including the HUB. The thermostat and humidistat system will lie directly in the room and components such as the temperature/humidity sensor will be visible to the viewer. These components will also be hard-wired to the room and hub. And the hub will still communicate with the application wirelessly.

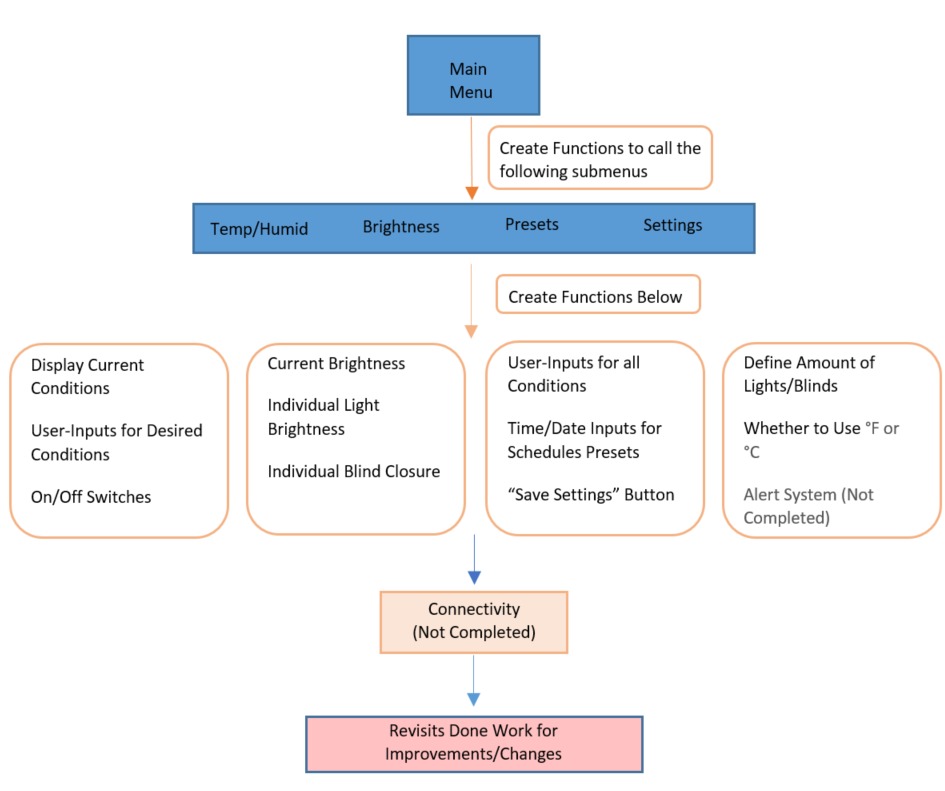
**Table 1. Team Members and Responsibility**

|  |  |  |
| --- | --- | --- |
| **Member’s Name** | **Color** | **Responsibility** |
| Chi Shing Poon | RED | Application |
| Muhammed Khan | BLUE | Hub |
| Yousuf Khan | BLUE | Hub |
| Alejandro Valencia | PURPLE | Control Room and Hardware |
| David Sigala | GREEN | Temperature and Humidity System |

## Application (Team member: Chi Shing Poon)

There are many softwares out there for developers to develop their Android application: MIT App Inventor, Eclipse, etc. For this project, the app was fully written in Android Studio using Java. The design progress included was five sections, which is illustrated in figure 2.

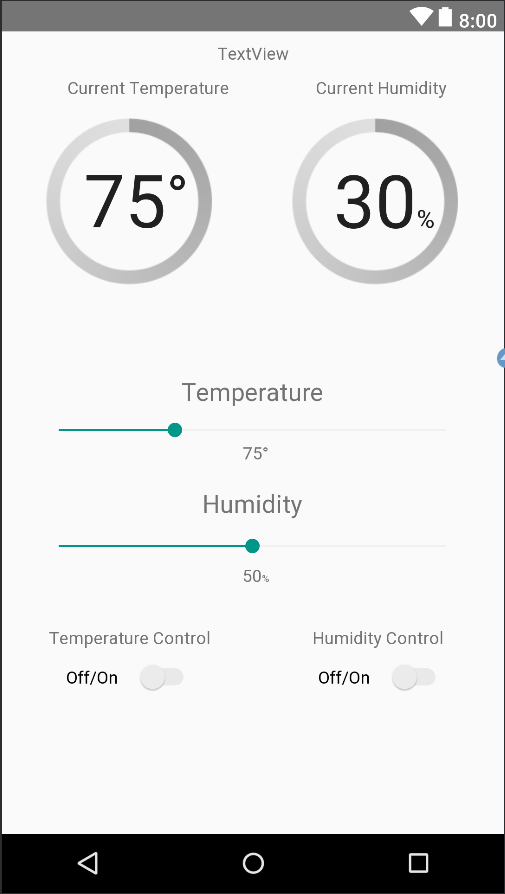
It began with deciding how many activities there are. Activities are another term for the different screens an Android application has. Then each screen would have a different design layouts; all of them have various input and output attributes. Upon finishing those layouts, functions had to be added to each of those attributes based on their purpose. Connectivity follows, which is via Bluetooth. Finally, we revisit everything to make any changes and improvements.

****

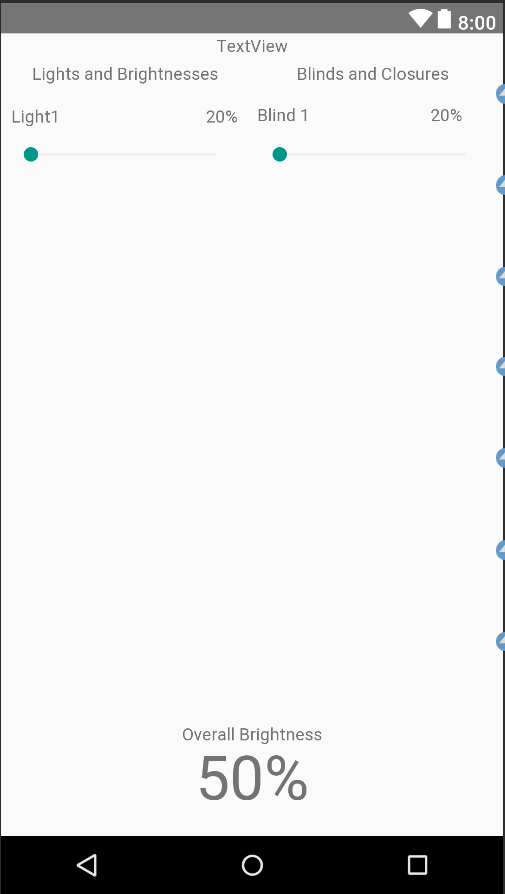
**Figure 3. Original Application Workflow.**

In our case, the applications have various activities serving different purposes: Main Menu, Temperature and Humidity, Brightness, Presets, and Settings. The Temperature and Humidity Submenu, as well as the Brightness Submenu, both have the same core purpose: To display the current condition in the room and to allow user input their desired conditions, in regards to their submenu topic (temperature, humidity, or brightness).

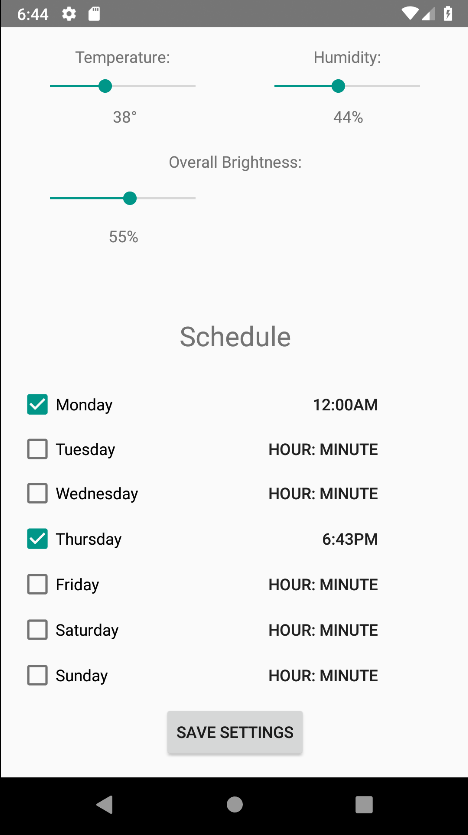
The way current conditions are displayed is through attributes called TextViews, these are simply text boxes that display a string value within them. For input attributes, slidebars were implemented, where users can freely adjust the bar in order to set their desired magnitudes for each condition. Each attributes are clearly labeled to provide the user a more intuitive experience. Figure 4 and figure 5 illustrates the submenu for Temperature and Humidity, as well as Brightness.

****

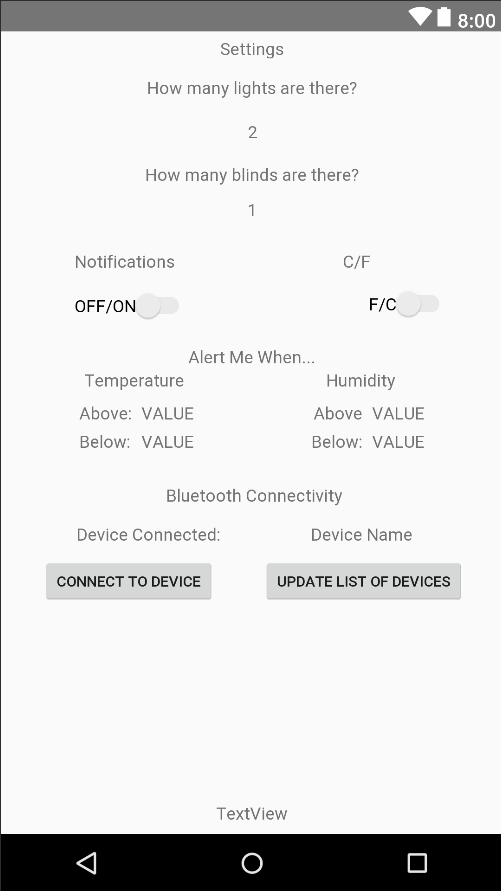
**Figure 4. Temperature and Humidity submenu layout.**

****

**Figure 5. Brightness submenu layout.**

****

**Figure 6. Preset submenu layout.**

****

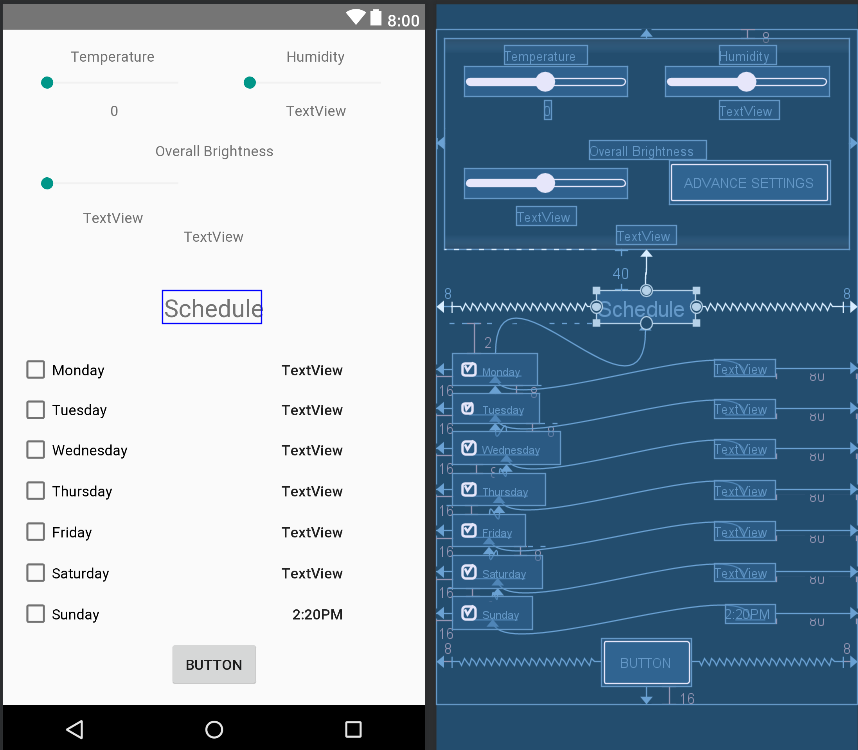
**Figure 7. Settings submenu layout**.

For the preset submenu, there are two parts of the user-input interface. First part is the top section of the activity, shown in figure 6, above the subtitle “Schedule”. This is where the user input their desired conditions for the room, which includes temperature, humidity, and brightness. Preferably, the brightness section would have individual controls over each lights and blinds; but since our group decided to control both lights together, only one input attribute was added for both lights. On the bottom section of the activity lies the user interface for a schedule. This is where the user can input when they want this preset to turn on.

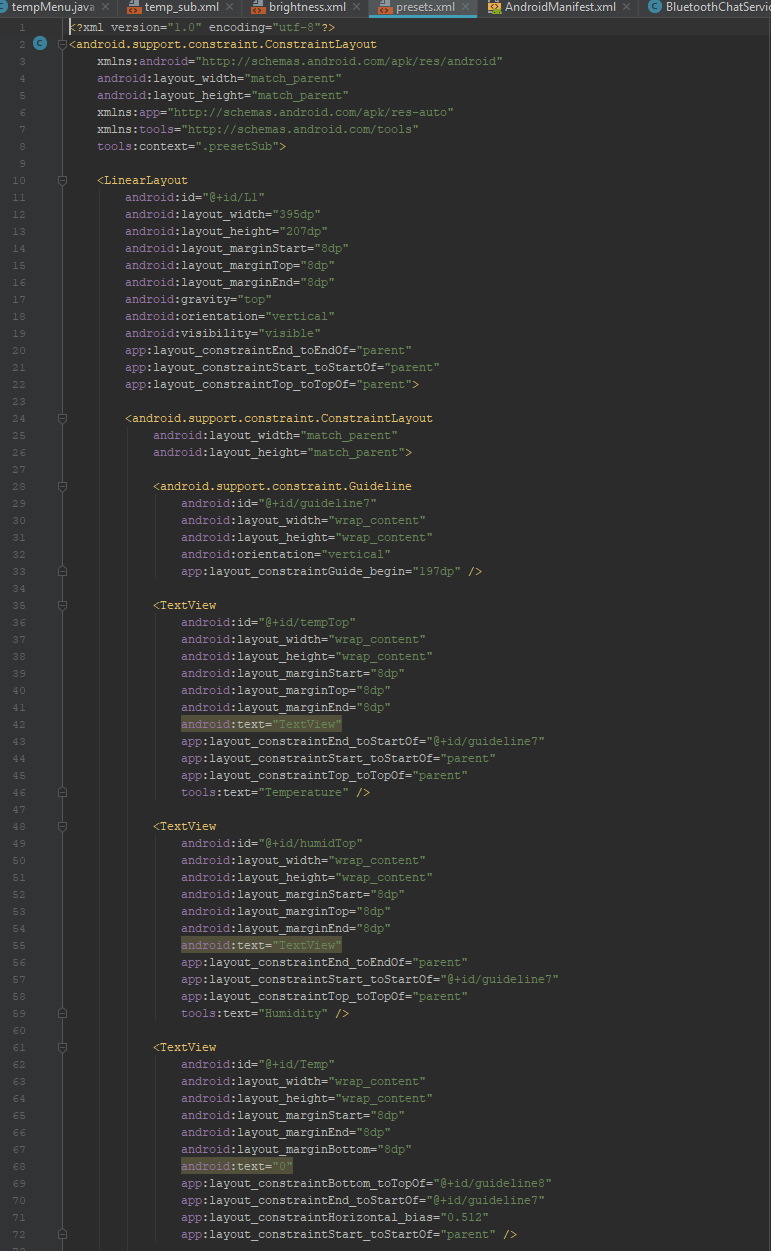
Since we currently do not have a function from the hub to detect how many lights and blinds there are, two simple prompts are placed in the setting screens. As shown in figure 7, there are questions asking the user how many lights or how many blinds there are. It also allows users to turn on and off the alert system, and to choose whether to have Fahrenheit or Celsius as the unit for temperature. In order to know when the users want to be alerted, there are input attributes for the users to customize at what temperature or humidity they want to be alerted. Below the alert section, is where the user can see what bluetooth device he connected to.

Currently, the setting activity is not completed, as the Bluetooth connectivity takes a higher priority. Since the alert system won’t be applied if there is no data received from the hub. Even though the connectivity isn’t completed, each activity is still being revisited for improvements.

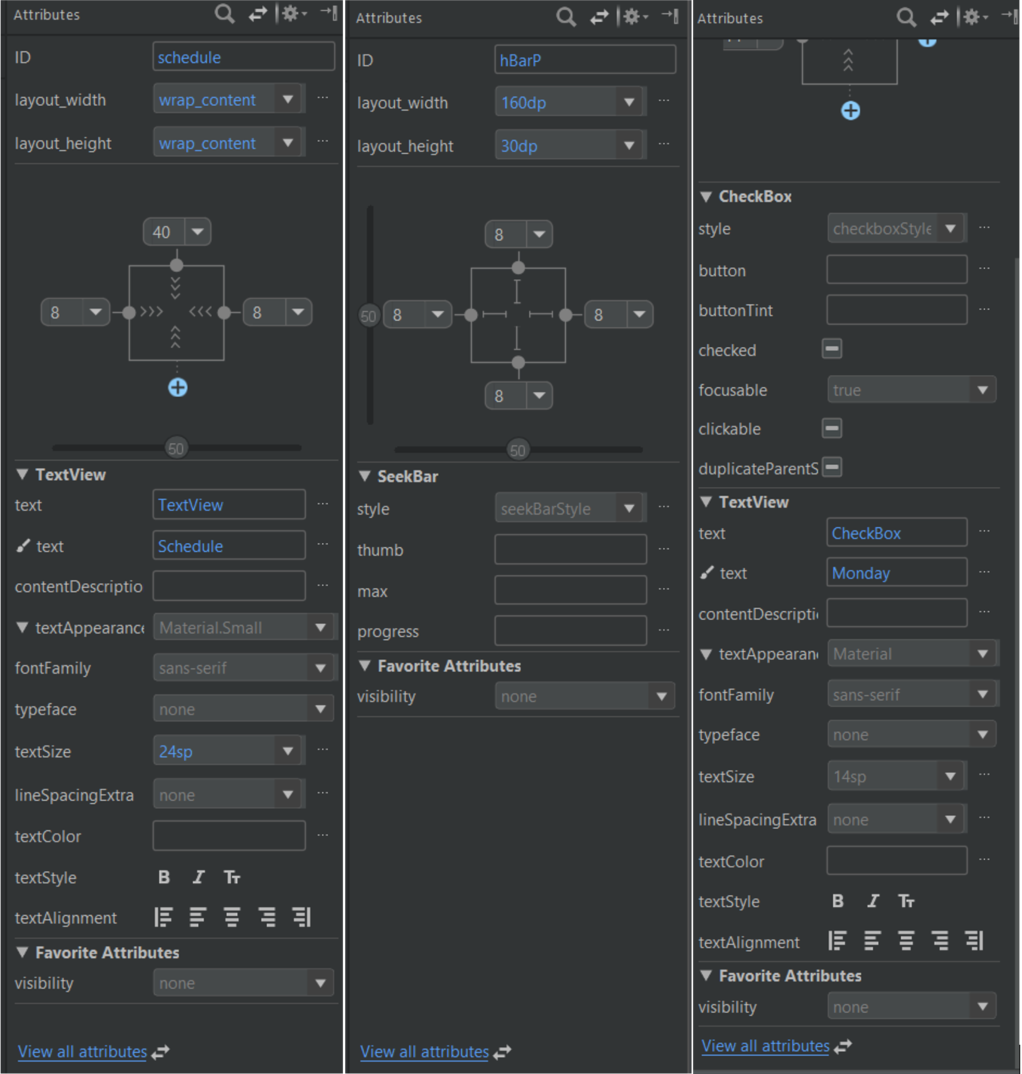
To go further into the development process, below are several figures from Android Studio. Figure 8a is the layout designer, it’s where the developer can place various attributes to design each activity layouts. They can also code these attributes into the layout via .xml files if they so chooses. Shown in figure 8b. The attributes then have properties like their ID, their text values, their dimensions, as well as positions, and more; shown in figure 9.

****

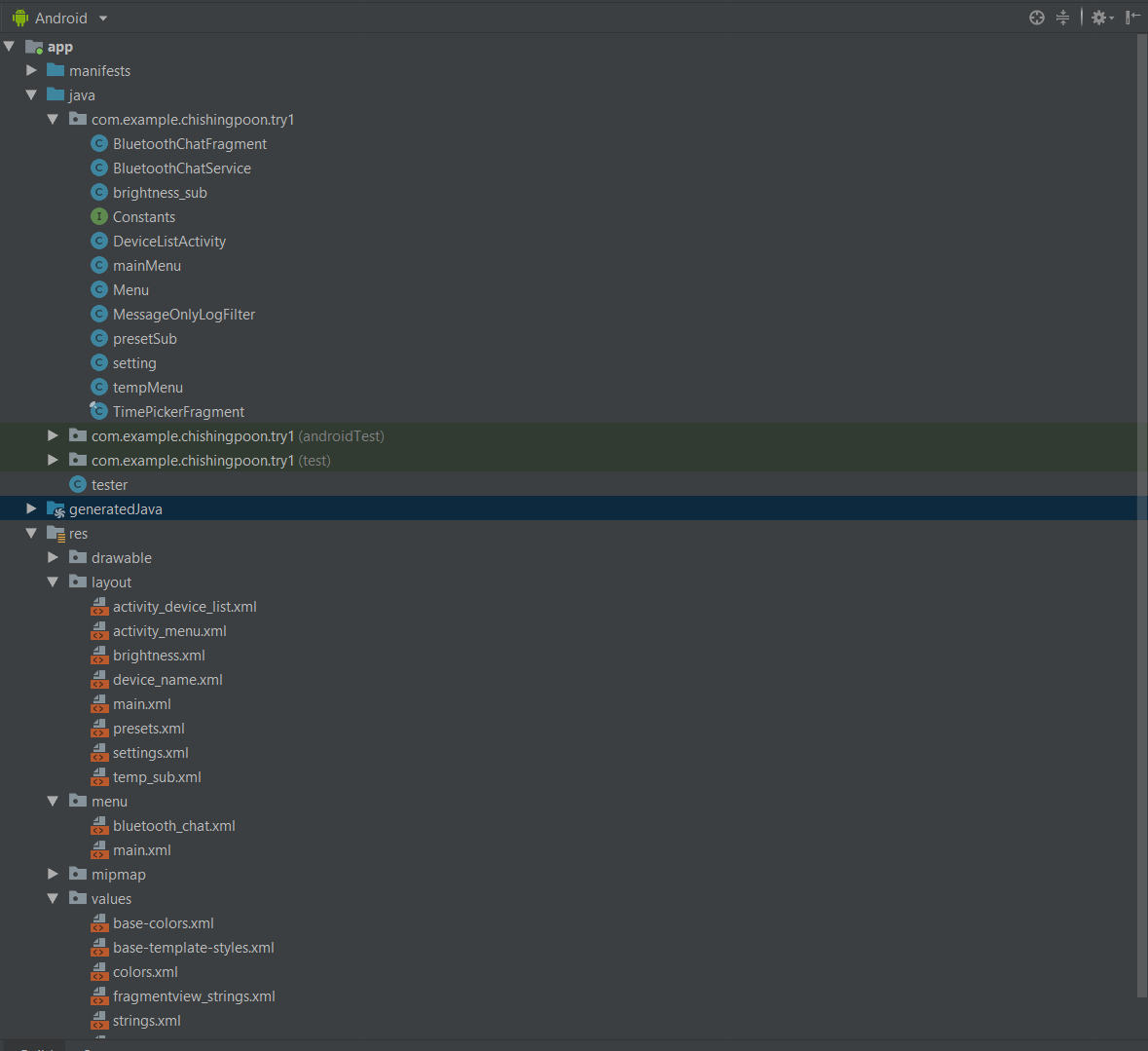
**Figure 8a. Layout Designer. (From Preset Submenu.)**

****

**Figure 8b. .xml files for layouts. (From Preset Submenu.)**

****

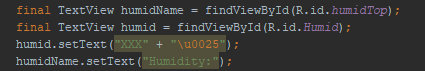
**Figure 9. Properties of different attributes. (From Left to Right: TextView, SeekBar (Slidebar), Checkboxes)**

****

**Figure 10. Project tree of this application.**

Designing an app can have an overwhelming amount of files to keep track of, these files are all shown in the project tree. Developers can freely create any folders to keep their files organized or create any files for the application to utilize. There are two types of files shown in figure 10, the .xml files and the .class files. The .xml files are the layouts shown earlier and the .class files is where most of the codes are. Which brings us into creating functions for attributes, and more. Since there are thousands of lines of codes in this application, this section will simply go over functions that are used multiple times or have a significant impact on the purpose of this application.

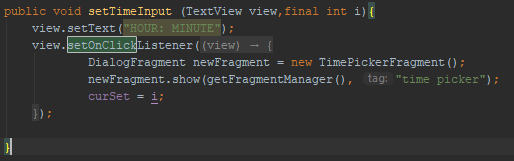
One of the mostly used attributes in this application was TextView. As mentioned earlier, it is basically a text box; it displays a string value. That is both a perfect output attribute, as well as an input attribute. For it to be an output, it is coded with a simple method of “setText();” with the string value inside the parenthesis. This method can be used for any attributes that can display a string value. But before they are allowed to be set a value, or any methods, they have to be defined as a variable first.

****

**Figure 11. Example code of setText(); for TextView.**

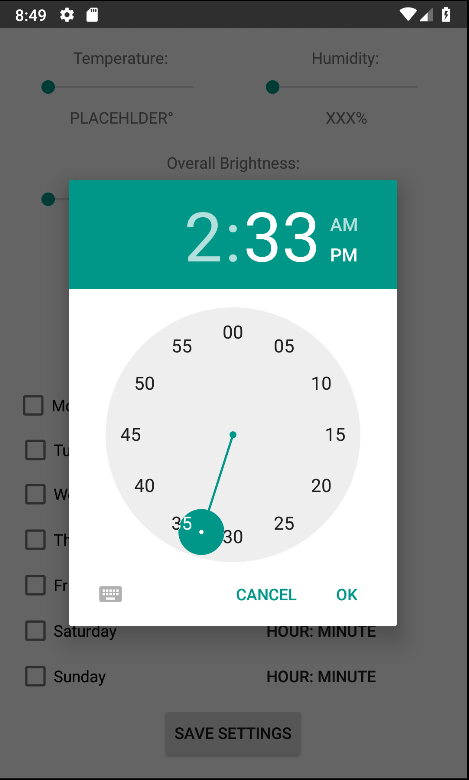
In figure 11, the first line is where a new TextView variable was created and “linked”. Each TextView in the layout design have an ID, and by linking that ID to a TextView variable in the .class file. You are able to change the different properties of that TextView, as well as deciding on what happens when certain action is taken. By using the method “setText();”, in figure 11, to the TextView variable humid (linked to the ‘humidTop’ attribute); it now has a value of “XXX%” with “\u0025” being the unicode representation for ‘%’.[4] Since Android Studio does not take most symbols ,percentage sign in this case, as a string value, but it does take unicode. So for most symbols would need to be represented by a unicode if they are needed in a string.

For it to be an input attribute, say the user can interact with a TextView placed on the activity. the method “setOnClickListener()” is used. This method has an inner nest of codes that will trigger when the TextView is clicked.

****

**Figure 12. Example code of setOnClickListener();**

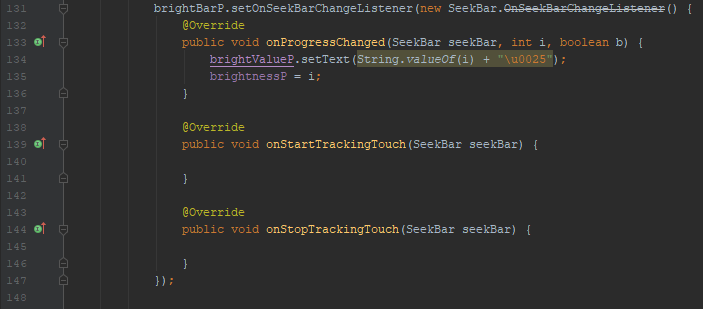
Above is the figure 12, it is an example of a method “setTimeInput”. Inside this method is an example of “setOnClickListener();”, where it calls a new object class (DialogFragment) in the fourth line and displays that object in the fifth line. Finally setting a variable to i. This code was from the preset submenu, it is called when user clicks on any of the TextView labeled “HOUR: MINUTE” from figure 6. The code triggered will then display a time picker, shown in figure 13, allowing the user to pick a time.

****

**Figure 13. Time Picker displayed using TimePickerFragment.**

****

**Figure 14a. SeekBar.**

****

**Figure 14b. Example code for “setOnSeekBarChangeListener();”.**

There are many different attributes in Android Studio that allow users to input values. For different attributes, they might have a different method that has the same function as “setOnClickListener();”. For example, seek bars, also known as slide bars, would use “setOnSeekBarChangeListener();”. Whatever code is inside its inner nest, would trigger when there is a change to the seek bar.

As mentioned above, there are many different attributes that allow the user to input values. So naturally, creating an application for the user to interact with would involve many of these “setOn\_XXX\_Listener();” methods. These methods were used repeatedly in this application for attributes like TextViews, SeekBars, ProgressBars, CheckBoxes, Switches, and Buttons.

Upon assigning each attributes with their methods, or functions, the development cycle moved into the connectivity portion. This is also the “development hell” of the development cycle. Connectivity was the main issue we had with this application, not to mention the entire project. Multiple solutions were researched for connectivity. First, for WiFi, online services like IBM BlueMix, ATT M2X, and Temboo, were considered the top solutions. These are all services that implements different APIs from various popular sites. API stands for application program interface, it is a combination of tools, communication protocols, and algorithms for building any programs.

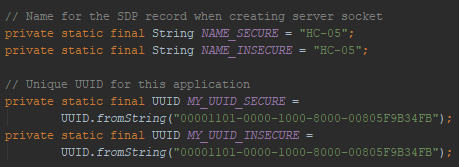
For example, Temboo had an API utilizing Google SpreadSheets. One of the solution was to use a MSP CC3220SF launchpad that will utilize this Google SpreadSheet API to read and write cells on a spreadsheet. The application would use that same API to read and write those cells as well. Which would’ve be the method of communication between the application and the hub if it had worked. Issues occurred as the CC3220SF wouldn’t run the codes for that API, which shut this option down. Multiple other issues aroused, which led us to change the direction towards Bluetooth instead of WiFi.

In order to communicate with the hub via Bluetooth, multiple tutorials on implementing Bluetooth were followed; but none had worked completely. Though they did serve some purpose as an introduction to some Bluetooth functions, like prompting users to turn on Bluetooth and viewing previously paired devices.

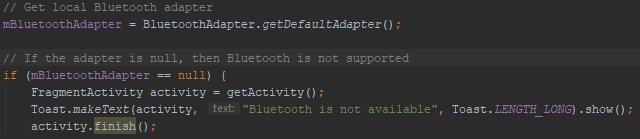
Finally, a sample code supplied by Android Developer[5],called “BluetoothChat”, was reviewed and tested. This code had its issues, which is discussed later; but with some small changes, it worked perfectly in communicating with the hub. Thus allowing the development cycle to progress.

Which is where the development cycle is at right now, after multiple delays, the BluetoothChat code is being implemented into the original app. In order to implement this code, a good understanding on how Bluetooth works had to be obtained. Using the BluetoothChat code, we understand that there are three main sections that the application needs to handle.

The first section is to prepare for the connection to another device. Each device connected to the phone via Bluetooth has a different UUID to them, UUID stands for universally unique identifier. It is a 128-bit number used to identify information in a digital system.[6] Naturally, the HC-05 Bluetooth module from the hub would have a UUID, which had to be coded into the BluetoothChat code as one of the changes made. After obtaining the UUID, the application would have to check if the device running this application has Bluetooth capability and then check if the device has Bluetooth turned on. Below are the code representations of those tasks

****

**Figure 15a. Implementing new UUID in the Code.**

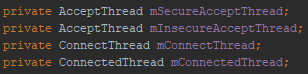
****

**Figure 15b. Codes to Check if Device is Bluetooth Compatible.**

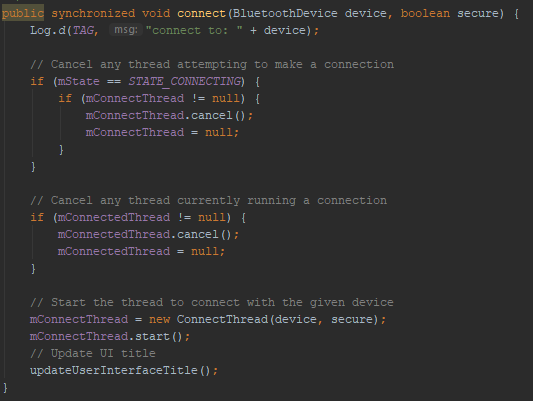
****

**Figure 15c. Checks if Bluetooth is enabled; if not, requests to enable.**

The second section is to actually connect to a Bluetooth device. This process needs the developer to create new threads to manage the Bluetooth connection. Typically, a very simple code would only have one thread running a time. But when a new thread is created, it will run at the same time as the original thread. So when you create a new thread that has the codes to allow the Bluetooth connection to begin, the original thread won’t be interrupted. Utilizing these threads in a method, the first thread would manage the initial connection, then the second thread would manage the connection while it is active.

****

**figure 16a. Threads created to be utilized.**

****

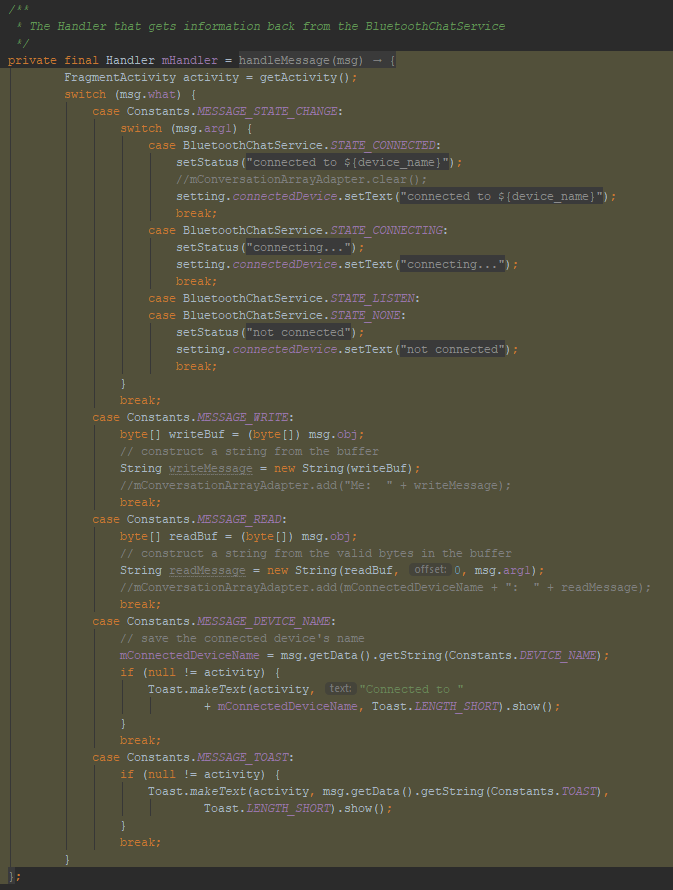
**Figure 16a. Codes to utilize mConnectThread in a function to manage initial connection.**

****

**Figure 16c. Code to utilize mConnectedThread in a method to manage connections while it is active.**

In figure 16a, new threads were created, then in figure 16b, the “connect” method manages the initial connection by stopping any connections or attempts at connecting if a new request to connect is made. In figure 16c, it manages active connections by stopping any connections that is running or any connections that isn’t used. In this case, it makes sure the device is only connected to one device. It also has a method in there that allows the device to send data out to the connected Bluetooth device.

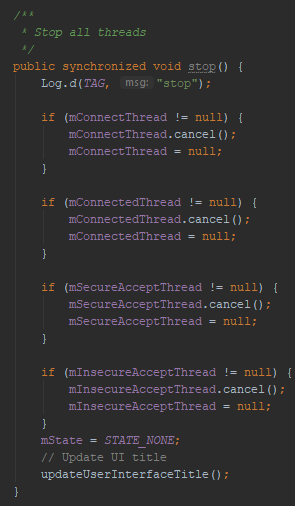
Which brings us to the third section, the methods that is called during the active connection to manage the data transceiving and receiving. Or to stop the connections.

****

**Figure 17a. A handler method to manage the data communications between devices.**

The handler method in figure 17a is a method that reads incoming messages one by one in bytes, then convert those bytes into a string and then the developer can decides what to do with that string. It also writes any messages the applications need to send out, by splitting up the string one by one into bytes and sending those to the other device. This handler method also manages various messages within the applications like displaying a string to notify the status of the connection.

Below is figure 17b, it illustrates another method that will stop all the threads when called. It is used when the connection has to be severed due to whatever reasons.

****

**Figure 17b. A stop method to stop any threads.**

Upon finishing the Bluetooth connectivity, the alert system would be implemented. Thus concluding the development process. As of now, the Bluetooth connectivity isn’t completed, but we have high hope in finishing the connectivity and have a working application on presentation day. If any difficulties shall occur and prevent that from happening, we plan to use a already-built application from the Google Play Store called the Serial Bluetooth Terminal. It is a simple serial terminal emulator that allows the user to communicate with a serial port device, which is what the HC-05 is.

There were many difficulties when developing this application, but only a few of them made a significant impact on the development cycle. Most of those difficulties were figuring out what each method take as parameter, or how to convert one variable type into another. Even though these issues are trivial on the surface, to new developers, these can be troubling. But Android Developer has a blog that explains and introduces every method in Android Studio, as well as sample codes that can be used to help troubleshoot applications with. [7]

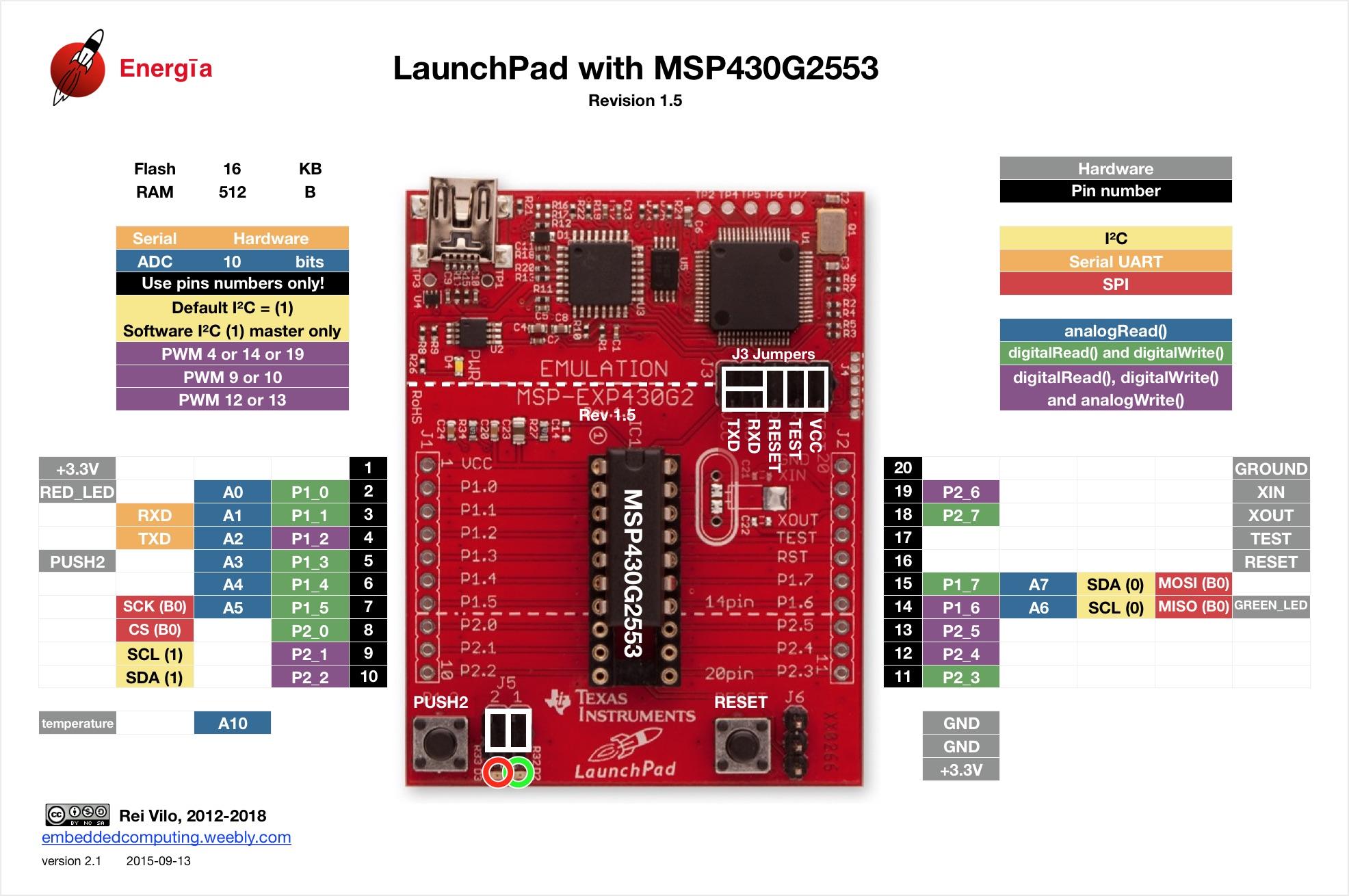
As to the difficulties that delayed the app over and over, was the connectivity issues. Most of the issues before this could be figured out by reading certain documents or browsing forums like StackOverflow. [8] But upon after browsing through many example codes and following many tutorials on Bluetooth connectivity, no progress was made as the attempt on implementing those codes resulted in errors after errors. Even the BluetoothChat sample code from Android Studio resulted in error as it wouldn’t even compile in the beginning. So a decision was made to migrate over to MIT app inventor in order to, possible, have a working application on presentation day. But a final attempt on Android Studio was made as we made various changes to the BluetoothChat sample code that allowed it to compile. The problem was that the original sample code was outdated, so some changes needed to be made in the code for it to run on a newer version of Android Studio.

So with the BluetoothChat sample code working, solving the one difficulty that restrained us from finishing the application; we began implementing its function into the original application and hope to finish it by presentation day.

## Hub (Muhammed Khan and Yousuf Khan)

During the beginning of this project we decided to go towards the ethernet/ CAT 5 route however, there was a slight hiccup on that because initially we thought that the CAT 5 was programmable and all we had to do was reprogram the microcontroller inside the nLight power pack [9]. This is basically an Ethernet type connection but the manufacturer said that it is not programmable. After realizing that we had concluded that CAT 5 and anything related to it will not be used in this project.

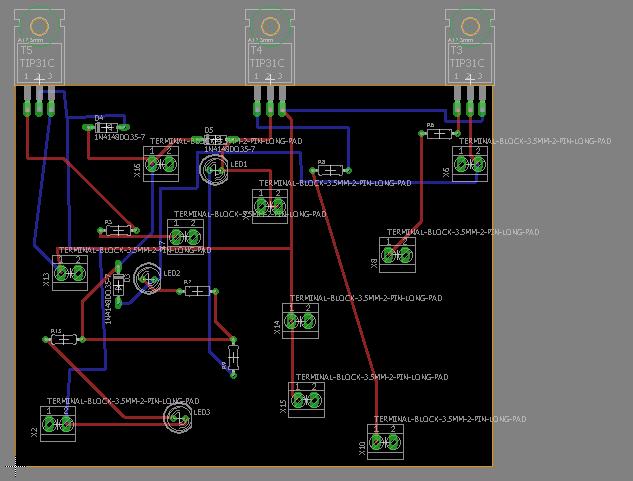
Wifi was then introduced into the project by using the CC3100 along with the MSP430 to try to see if it can be controlled through an active internet connection, this was achieved first by doing some extensive research on how we can get started because we did not have much experience with code and the examples we had come across were mostly arduino based and it did not meet our requirements. However, after some research there was a program called energia that takes the code from arduino and helps translate it into an MSP430 code after manually converting the pin out shown below.



**Figure 18. MSP430 pinout for Energia**

Next, we decided that the connection between the msp430 and the hub would be through the IBM Watson platform [10] that allowed the device to communicate to the server and this connection was successful after coding the msp430 along with the cc with some code to help it connect. However, the connection over wifi through the CC3100/ MSP430 was successful but the issue was that the Application our other group member was working on did not comply with the IBM watson cloud requirements which made it difficult for us to proceed. After successfully being able to communicate with the IBM watson Bluemix server the device was registering to the cloud at a constant rate of what the temperature in the room is and it constantly send this data over the internet every couple seconds. once again, it was very challenging trying to create an Application to process this data and send it out all in a short amount of time that was given. More details on the application will be provided under the Application section.

After a rough ride through both the CAT5 and the Wifi not properly functioning as anticipated. The group decided to go ahead and go with the Bluetooth route because that did not require as intense of an application as it did if the project continued with wifi. However, when doing the bluetooth a proper module was needed instead of the previous module used for wifi known as CC3100 this time a HC-05 module [11] was used to establish a connection through bluetooth and a simple application was used known as serial bluetooth terminal. this application was coded in order to just recieve simple commands and simple streams of data, all components of this part were successful in working using this application. However, this was not the final design for the application as the vision of the application was much more complex. Another key component to remember is that the MSP 430 is only allowed to send max 3.3 volts and the lights we are using require 120v so we ran the mps430 through a NPN BJT transistor and a relay allowing everything to be activated. the schematic was built in eagle and the relay is shown above.



**Figure 19. EAGLE connections and routing**

After the schematic was built in Eagle the PCB design was also routed and sent to milling on PCB board. The entire routing was done manually and ensured that there weren't any overlaps. This was by far the most challenging part of the project because it required intense rerouting ensuring no wires were overlapping. After multiple efforts it came out successfully and worked with our project. The next step was ensuring that the temperature sensor and humidity sensor was also configured into the original code of the MSP430 and HC-05 because originally the group decided to work separately on different parts and now we are making sure everything comes together and is properly aligned with everything making sure nothing is out of place. The way this was done by adding another function into the code for both humidity and temperature allowing it to function with the serial link application. The code was then used to make sure that it was compatible with the application using the MIT app inventor.

The hub will take commands from the application through serial data. For example, each command will be assigned to a character (number, letter or symbol) that will trigger an action in the HUB. Furthermore, to keep a level of simplicity, brightness, temperature and humidity will also behave in the same way. The user will be given a range of temperature that they can set, for this project it will be limited from 68-78 degrees fahrenheit. Further applications can include a wider range. Brightness will be controlled through PWM and will be arranged in intervals of 50 from 0-255. Lastly, the Humidity will range from 15-50% in intervals of 5. This will allow the user to use only realistic levels of humidity in their control. In more complex applications of this technology, a wider range and more precise levels can be applied for all three of these variables.

The code for temperature and humidity will act very similarly. Depending on the user’s input the variable for desired temperature or humidity will change within the microcontroller. That will be compared with the actual value and whether it is higher or lower, it will trigger the fans and/or humidifier. Though an activation for a heater was included in the code, the final design will not include an activation for the heater.

Data is sent back to the application using the “Serial.write” function. This way data about the current state of the room such as brightness temperature and humidity can be visible in the application.

## Control Room (Hardware) (Alejandro Valencia)

In Order to create the room, we first used a scaffold (6ft x 4ft) which has a sturdy base that can support up to 800 lbs. This is plenty for all of the load that will be placed on it. The reason we used a scaffold was to have a strong base and stands that could support the walls and roof that will be hung from above.



**Figure 20. Image of the scaffold that will be used**

After the scaffold was built, 2x4 (2ft x 4ft) planks where added around to the outer area of the scaffold. These 2x4 are used as an anchoring point for the walls that will be installed. Sheetrock was placed around on the base, Top, left, right, and back panels which will be demonstrated as a room. The rooms were then anchored down by using sheetrock screws. After this is complete, we started the wall- rough the lights, switch, fans, and a custom-built box for the LCD screen. One gang box where used to anchor onto the sheetrock. A small square cut (3ft x 3ft) was cut on the back panel to represent the window.



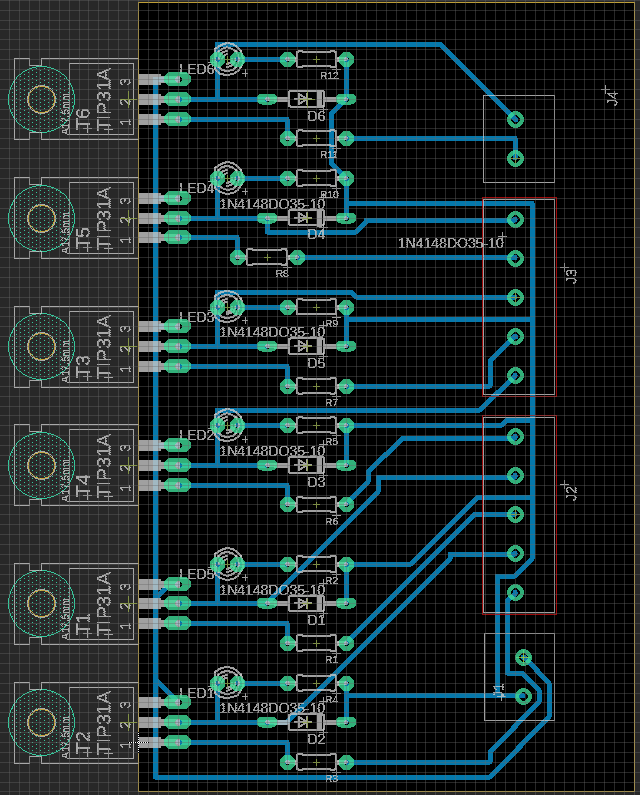
**Figure 21. Walls placed in the room**

The next procedure that was done was to start the 1st coat of paint. All corners were cover up with caulking and filling to cover up any imperfection that may have occurred. The same process would be repeated one more time to better results.

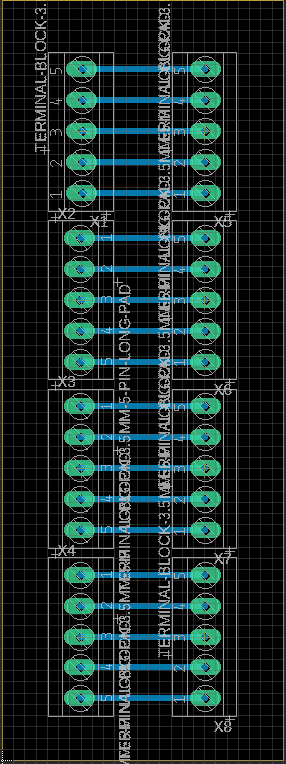


**Figure 22. Room all painted**

Once the paint was dried, we were able to install all devices such as the outlet, fans, switch, and LCD screen. Plexiglass was installed onto the cut that was used as window provide more authenticity to the room. To hold the plexiglass in place, trim was used on the perimeter to hold the plexiglass from falling. After all, this was complete wires where ran to the all the devices and they are in the process of being made up. A motor and blinds were added above the window to make the window more authentic. All wire is run into a box that is installed in the back of the room. The reason for this back is to make safe the room so that no one will be touching a live wire. Live wire means that there is current in the wire. At the moment the box is not made up due to that we are waiting for the PCB boards to be soldered.

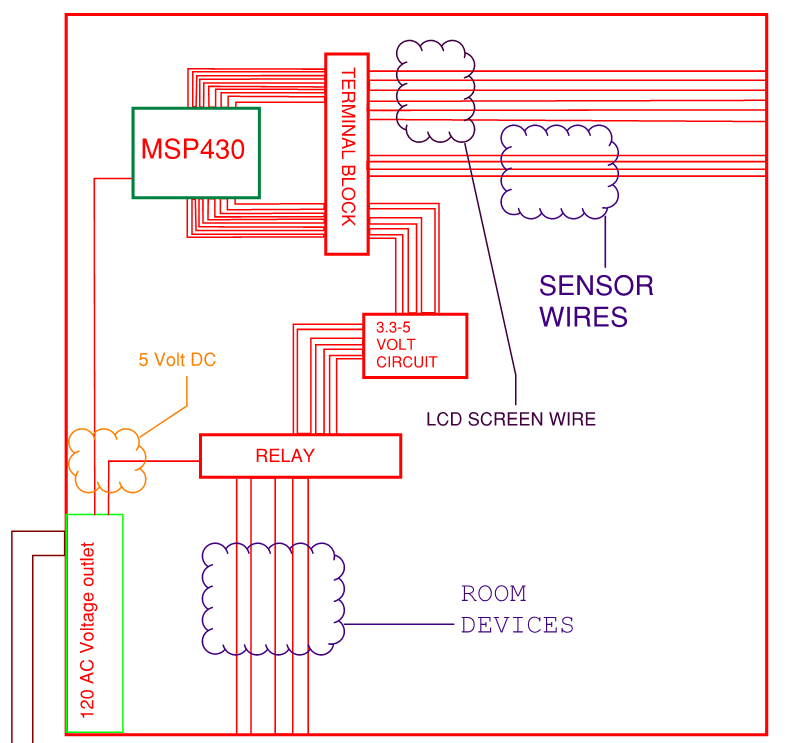


**Figure 23. PCB board what will act as a relay for the 3.3 to 5 Volts**



**Figure 24. PCB board design for a input and output to the MSP 430**

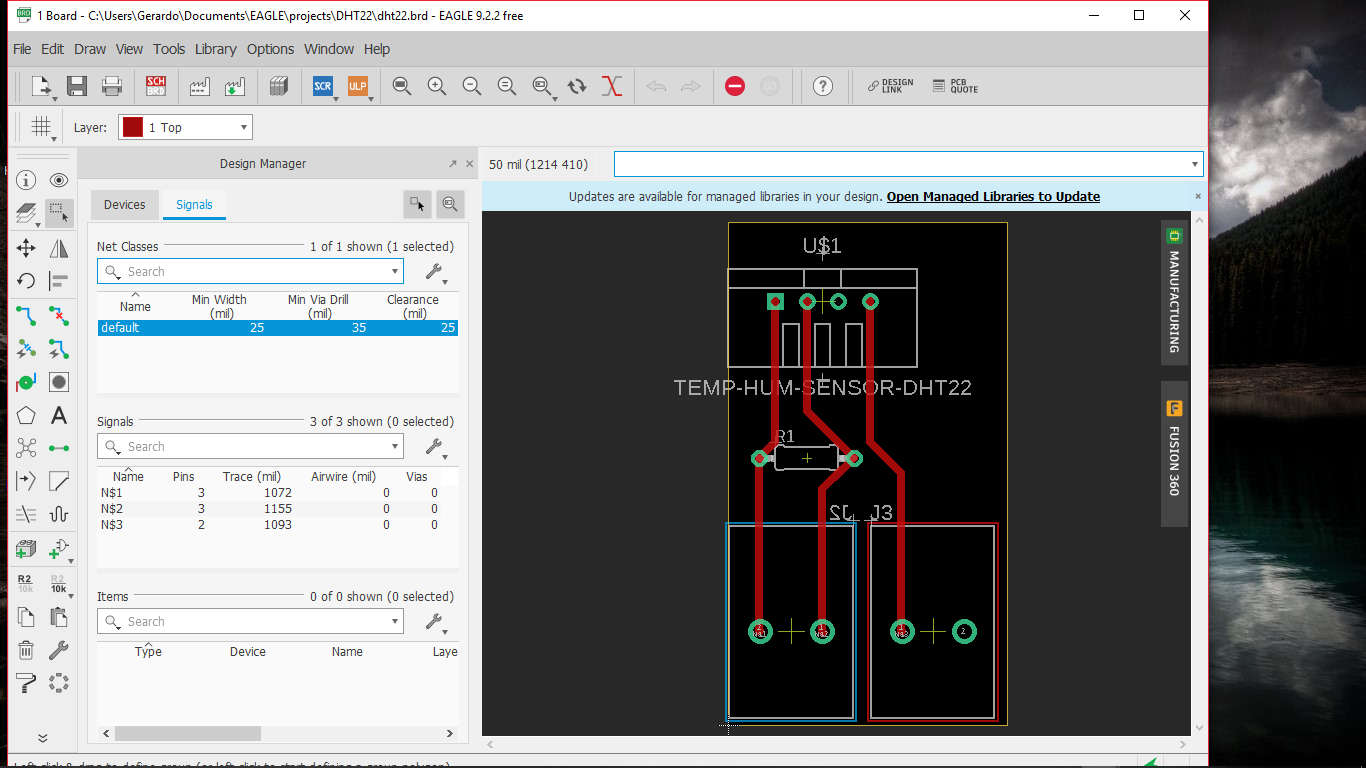
A photocell sensor will be placed at the center of the room to detect any light that is not related to the room. The photocell is not yet installed at the moment, we are waiting for the part to arrive so that it can be installed. Our previous sensor was accidentally cut. We decided to go with this large scale to make the space feel more like a room just a small box that could have been used by any other group. There was no expense to using that material used to create the room rather than the small devices like that LCD screen and the sensors. Small desk fans were used to demonstrate the A/C and Exhaust in the room to cool the room and remove any humidity in the room. The room is too big to demonstrate the room cooling so we first use a hair dryer to heat up the sensor which will show that the room is heating up in which will activate the fan to turn on. In order to demonstrate the removal of humidity, we will spray water onto the sensor which will detect that the room is too humid which will turn on the exhaust fan. To demonstrate the lights dimming in the room we will be using a flashlight that will be flashed on to the photocell which will dim the lights.



**Figure 25. Schematics for the Makesafe box**

## Temperature and Humidity System (David Sigala)

For the Temperature and Humidity system we chose ultimately to go with the DHT22 to measure the values, the UCareAir cool mist humidifier to add moisture to the room, and a system of fans for cooling and dehumidifying. The DHT22 is a digital humidity and temperature sensor that uses a Thermistor to gauge temperature, and capacitive elements to detect the humidity[12]. Thermistors are resistors whose resistance changes with temperature. Capacitive humidity sensors usually work by having a bit of metal oxide between electrodes. This metal oxide is sensitive to capacitive changes, so capacitive humidity sensors measure the difference percentage wise for relative humidity[13]. These features make the DHT22 accurate enough. This sensor only uses three of its four pins VCC, ground, and the data input. To ensure accuracy, a pull up resistor of 4.7k-10k ohms is used between the VCC and data input pin.



**Figure 26. EAGLE connections and routing for sensor**

In the end we chose to go without the LCD screen. Although we had it correctly configured to display the temperature and humidity, due to pin limits on the MSP430 and its admittedly redundant nature with respect to the app, we chose to scrap it.

We used the UCareAir cool mist humidifier for humidifying the room. However, the activation of the device via button proved beyond our level of expertise. Instead we opted to to approach this from a different angle. Instead of turning the device on and off, we could leave it on the whole time and instead control its flow. While a solenoid valve would have been ideal, the pressure and current requirements were higher than even our relays would be capable of reaching without burning out. We opted to control the flow of the mist with a small 1” fan. This fan would be always on to prevent the always on humidifier from humidifying the room. When the room needs humidity the fan will turn off. To make sure the mist wasn’t shot out stronger than the fan, we sent it through a few inches of 1 ⅛ “ PVC pipe with two openings. One leads to outside of the room, and the other inside.

For cooling the room we used a system of relays to power on fans. The plan is to have these fans turn on when the temperature detected is greater than the desired temperature. Likewise, when the Humidity is less than the desired, the pipe fan will turn off. When the Humidity is greater than what the desired humidity is, the exhaust fan (modeled similarly to the cooling fan) will activate pushing the moist air out of the room to be replaced by dryer air. We had a plan to make a heating element, but because of fire hazards and massive power consumption we had to scrap it.

# Decision on Solution and Part Selection

**Table 2. Part Selection for Hub and Application**

|  |  |  |
| --- | --- | --- |
| **Item** | **Price ($)** | **Description** |
| **MSP430G2559** | **0** | **Microcontroller** |
| **HC-05** | **0** | **Bluetooth Module** |
| **USB-C** | **0** | **Connector Cable** |
| **Google Pixel** | **0** | **Application Development** |
| **Google PIxel 2** | **0** | **Application Development** |
| **Jumper Cables** | **0** | **Connectivity** |
| **8- Channel Relay switch** | **9** | **Control** |
| **NPN BJT Transistors** | **0** | **Control** |
| **Resistors** | **0** | **Control** |
| **Diodes** | **0** | **Control** |
| **LED’s** | **0** | **Control** |

One of the main reasons the MSP430G2 was chosen because of its compatibility with the selected bluetooth module. The bluetooth module being mentioned is the HC-05. This device allows for easy connectivity and pairing with all bluetooth enabled devices. Simple jumper cables were used connect the microcontroller to devices such as the relay. Android application development was chosen because of its friendliness to developers versus iphone development. The application will be run on both the Google Pixel and Pixel 2 because they are the ideal devices for Android applications and they run the most pure, stock version of Android. These devices use USB-C cables to load the applications onto them so that was also needed. All of these components for these sections besides the relays are already owned by members of this project or were retrieved from the university so the net cost for this portion would be the price of the relay.

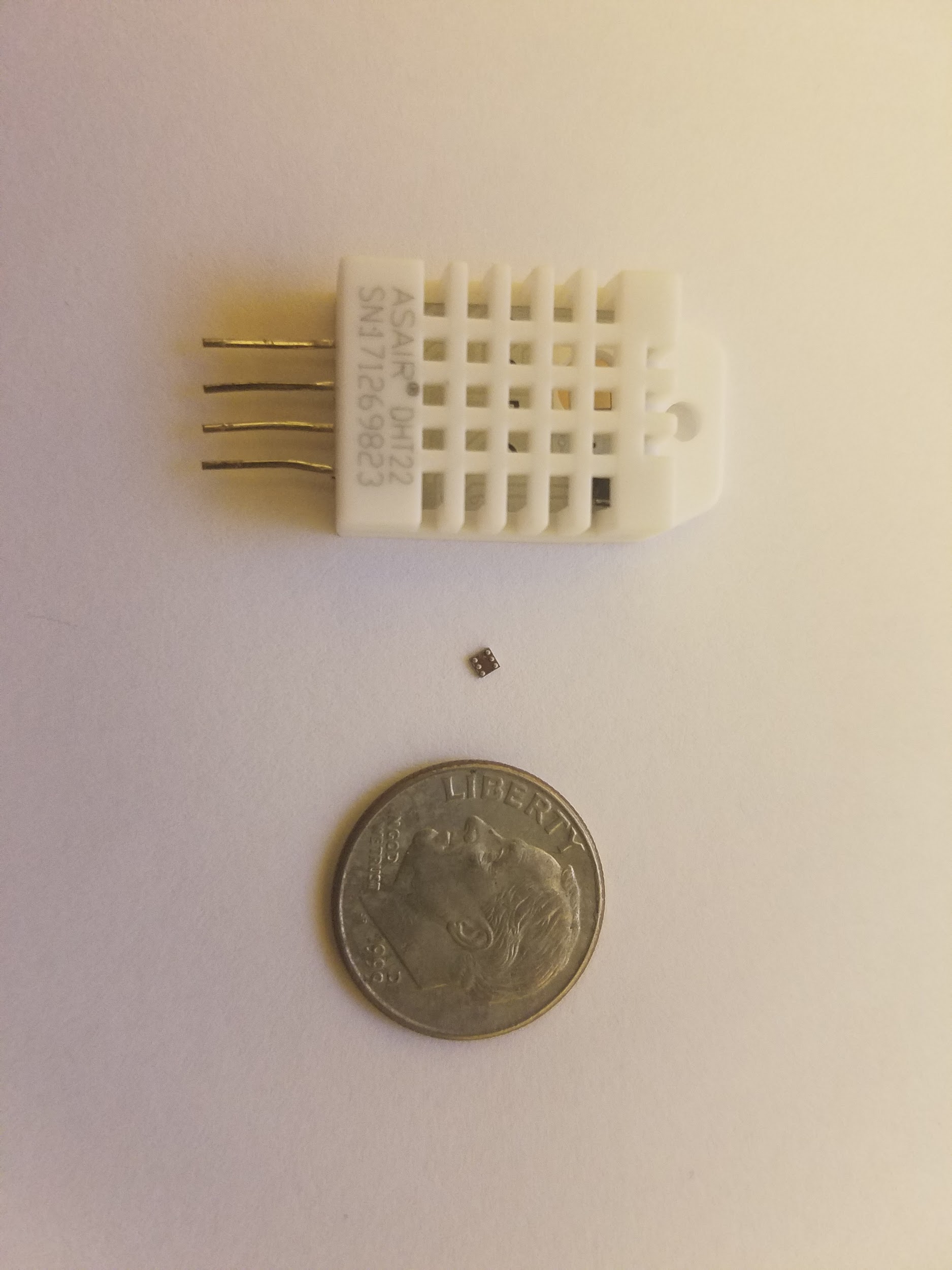
The relays that were used can only be controlled by a minimum of 5v. Our microcontroller only has the capacity to output a maximum of 3.3V at the high setting. To control the relays, a circuit that uses an NPN BJT transistor with a 5v source tied to it was used. The 3.3V source from the msp430 would control the transistor which, when on, would allow the 5v to activate the relays. This circuit included an NPN BJT transistor, resistors, LED’s, diodes and a 5V source. The schematic in EAGLE before being milled on a PCB board can be found in the HUB section of this report.

**Table 3. Components for the Control Room**

|  |  |  |
| --- | --- | --- |
| **Material type** | **Price ($)** | **Description** |
| **Scaffold** | **0** | **Skeleton** |
| **4x8 sheetrock** | **0** | **Skin** |
| **Joint Compound** | **0** | **Plaster** |
| **White paint** | **0** | **Cosmetics** |
| **12/2 Romex** | **0** | **Power** |
| **18/2 Low voltage wire** | **0** | **Dimming** |
| **Screen Mesh** | **0** | **Blinds** |
| **DC Motor** | **2** | **Motor** |
| **2 USB Desk fans** | **14** | **Fans/Exhaust** |
| **2 computer fans** | **20** | **Cooling** |
| **2 pole/Dimming switch** | **0** | **Switch** |
| **Can Lights** | **0** | **Lights** |
| **Impact screw gun** | **0** | **Tools** |
| **LCD screen** | **11** | **Thermostat/**  **Humidistat System** |
| **DHT22 Sensor** | **10** | **Thermostat/**  **Humidistat System** |
| **Cool mist humidifier** | **0** | **Thermostat/**  **Humidistat System** |
| **Total Cost:** | **64** |  |

`The hardware and devices showcased in Table 3 amount to the totality of what we aimed to control. These components make up the temperature/humidity system, as well as the light system, and the room itself. The components that show a price tag of “0” are there because of personal contributions and possessions. Some of these were already owned beforehand and are being temporarily used for the sake of the project.

Originally we were planning to use the HDC2010YPAR temperature and humidity sensor, however as can be seen in the figure, the size proved to be much more difficult to work with than we were intending.



**Figure 10. Comparison of two sensors with a dime. (from left to right) DHT22, HDC2010YPAR, dime.**

This led us as a group to decide on choosing a more user friendly sensor, the DHT22.The use of a thermistor and capacitive humidity sensing makes this sensor accurate enough for the scope of a room[12]. Another reason this sensor was chosen was for the relative abundance of sample programs. Since the sensor takes in small analog inputs and performs digital conversions with a small chip inside of it, it would take many different precise timing programs and diagrams to correctly configure it[14]. By getting the correct libraries, it is easy to read the temperature and humidity values[15]. For humidity, we chose to go with the UCareAir Cool Mist Humidifier. Normally the cool mist can launch mist on average to 23”, which if the sales pitch is to be believe is better than the average humidifier[16]. The reason we chose a 1” fan was because we planed to use pipe to slow down the flow of mist and work off of our already working fan controls. The pipe was picked because of its approximate size to the mist output hole of the UCareAir cool mist humidifier.

In the room we have a couple of USB fans that were chosen for their surprisingly large size in reference to their power requirements. These fans were able to generate enough force to push the air towards the required locations. With the relay system we can power each of these using 5 volts, and still get cooling into the room. Can Lights were picked because of their availability and incredible ability to light up the room. The two-pole dimming switch was needed for adjusting the non-natural light.

# Results

Wifi connectivity was initially chosen over bluetooth because it is understandable that the user will always have their WIFI enabled on their phone. However after countless efforts, it was concluded that connecting through wifi would not work out. That is when it was settled that that the final result would include bluetooth connectivity. Though this solution was our only option,it became apparent that it would be more beneficial to go this route.

Firstly, it is known that a bluetooth connection is more reliable than a WIFI one. Secondly, this gave us the opportunity to pivot our customer audience towards the commercial world. This would include businesses and factories. Lastly, using a bluetooth module would just be a plug and play ordeal rather an having to flash, code and introduce libraries for the chosen booster pack. However, the drawback that was introduced with using bluetooth was the compatibility with the application. This is unavoidable and it is also unknown if such issues would have also been faced with WIFI.

1. Conclusion

Despite all the decisions and the pivots made in this project. Students learned a tremendous amount of knowledge. The HUB that was stated earlier went through tremendous amount of research and exploration allowing the Project to get to where it's at today. Some of the things that can be taken away from the HUB would be the way students learned to communicate between the different types of devices simultaneously through a Web server whether that being Amazon Web Services or IBM bluemix server which this project is primarily focused on. Unfortunately, some of the things did not go as desired as it was thought out, one being that connectivity could not be established between one of the devices, causing the primary focus of this Project to divert towards Bluetooth connectivity. However, given more time it could have been possible for it to work.

Bluetooth was made easier in terms of connectivity. However, the application also had to be changed in order for it to meet the standards of bluetooth. Bluetooth is fast and simple to set up, so simple that it requires the user just to have a device with bluetooth connection, no paid internet fees, no bandwidth issues. This creates more security and more reasonable expenses in regards to the user.

When it comes to the temperature and humidity system, we were able to figure out how to get the fans and by extension humidity system to work. We ultimately had to scrap the heating due to safety concerns, and the humidifier because of room implementation problems. We are confident that the system for cooling, humidifying, and dehumidifying will work in conjunction with the HUB. While the system seems like it would end up running forever with fans turning on and off constantly, in reality this is all normal. Most temperature systems are meant to be constantly on, constantly correcting for changes in the environment to assure maximum average comfort to the recipients.

Control of the devices was tested and verified through a bluetooth serial monitor app. This verified that all of the functions are working as normal. However, because the pivot to bluetooth was made too late functionality of the app that was being developed had suffered. The layout for the app was perfected, but the communication portion has still not been completed at the time of this report. However, we are hopeful that this can be completed by the presentation date. If it fails to work on the presentation date, the room can still be controlled through either a serial terminal emulator acquired from the Google Play store or the bluetooth chat app which referenced in the app section of this report.

1. Statement of Future Work

In order to complete this design, connectivity between the hub and the created app needs to be established. Initially, the design accounted for a connection through WIFI. After many failures, it was decided that a pivot towards bluetooth connectivity would be necessary. As a result, the application suffered because of the difficulty of implementing bluetooth connectivity to an android application.

Despite countless attempts, and sifting through mountains of sample code, providing bluetooth data transfer between the bluetooth module and the android application has failed. Many attempts have ended in either no data being sent or the application crashing. As previously stated, the room is able to be controlled through a simple communication method. To complete the design, This functionality needs to be added to the existing application to complete the design.

Currently, settings that are offered do include ranges that the typical user would typically want for their temperature and humidity. To further improve the viability of the design, a wider range of settings could also be added. As for brightness, more accurate settings could be offered. Sometimes people want their brightness to be at an exact value and our current design only offers twenty percent intervals.

##### References

1. Rotman, D. (2017, May 22). Here's how to use AI to make America great again. Retrieved September 26, 2018, from <https://www.technologyreview.com/s/603465/the-relentless-pace-of-automation/>
2. Logitech Study Shows Multiple Remote Controls Hindering Entertainment Experiences Around the Globe. (n.d.). Retrieved September 26, 2018, from <https://www.logitech.com/en-us/press/press-releases/7748>
3. Stella, R. (2017, January 28). Think your hotel room's heater is ignoring you? You're probably right. Retrieved September 26, 2018, from <https://www.digitaltrends.com/home/hotel-thermostats-rigged/>
4. (n.d.). Retrieved December 2, 2018, from <https://www.rapidtables.com/code/text/unicode-characters.html>
5. Googlesamples. (2018, February 16). Googlesamples/android-BluetoothChat. Retrieved December 2, 2018, from <https://github.com/googlesamples/android-BluetoothChat/#readme>
6. Universally unique identifier. (2018, November 11). Retrieved December 2, 2018, from <https://en.wikipedia.org/wiki/Universally_unique_identifier>
7. Developer Guides | Android Developers. (n.d.). Retrieved December 2, 2018, from <https://developer.android.com/guide/>
8. Where Developers Learn, Share, & Build Careers. (n.d.). Retrieved December 2, 2018, from <https://stackoverflow.com/>
9. <https://www.acuitybrands.com/products/detail/147279/nLight/nPP16-Family/-/media/products/nLight/147279/document/NPP16-EFP_pdf.pdf?DOC_Type=SPEC_SHEET>
10. Two-factor authentication. (n.d.). Retrieved December 2, 2018, from <https://console.bluemix.net/developer/watson/dashboard>
11. HC-05 Bluetooth Module Interfacing with MSP-EXP430G2 TI Launchpad.. (n.d.). Retrieved December 2, 2018, from <https://www.electronicwings.com/ti-launchpad/hc-05-bluetooth-module-interfacing-with-msp-exp430g2-ti-launchpad>
12. Adafruit Industries. (n.d.). DHT22 temperature-humidity sensor extras. Retrieved October 30, 2018, from <https://www.adafruit.com/product/385>
13. Humidity Sensor: Basics, Usage, Parameters and Applications. (2018, January 04). Retrieved December 2, 2018, from <https://electronicsforu.com/resources/electronics-components/humidity-sensor-basic-usage-parameter>
14. DHT11 & DHT22 Sensor Temperature and Humidity Tutorial. (2018, October 21). Retrieved December 2, 2018, from <https://howtomechatronics.com/tutorials/arduino/dht11-dht22-sensors-temperature-and-humidity-tutorial-using-arduino/>
15. JFF-Bohdan. (2015, April 13). JFF-Bohdan/dht22. Retrieved December 2, 2018, from <https://github.com/JFF-Bohdan/dht22>
16. Cool Mist Humidifier - Super Quiet Humidifier for Baby Office, All Night Moisture Ultrasonic Air Humidifier for Bedroom, Powerful Mist, NO Wet Floor, Easy Use & Clean, Auto Shutoff Portable Humidifier. (n.d.). Retrieved October 30, 2018, from <https://www.amazon.com/Cool-Mist-Humidifier-Moisture-Ultrasonic/dp/B07C1Q679J>

**APPENDIX**

Gantt Chart:

